**Hard Programs**

1. Ganesh Reddy

192110023

21.

#include <iostream>

int fibonacci(int n) {

if (n <= 0)

return 0;

else if (n == 1)

return 1;

else

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

int n;

std::cout << "Enter the number of terms for Fibonacci sequence: ";

std::cin >> n;

std::cout << "Fibonacci sequence up to " << n << " terms:\n";

for (int i = 0; i < n; ++i) {

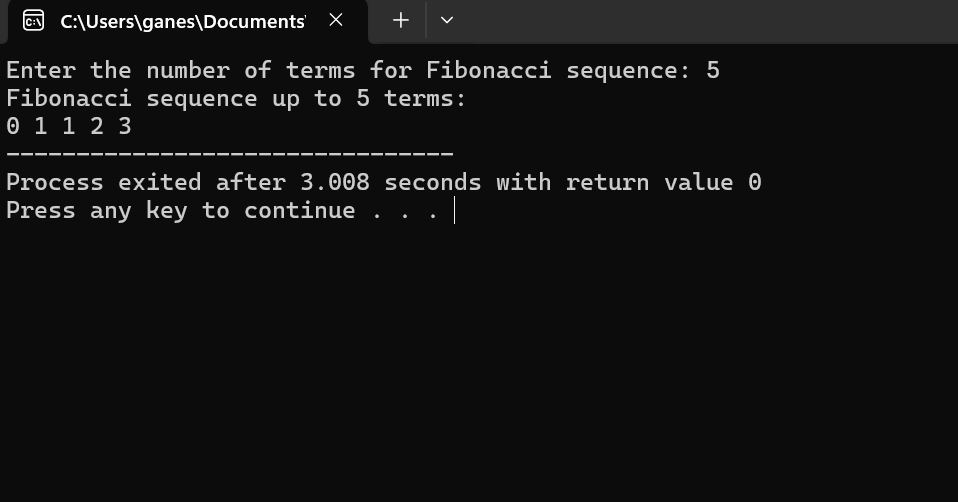
std::cout << fibonacci(i) << " ";

}

//std::cout << std::endl;

return 0;

}



22.

#include <iostream>

#include <cstdlib>

using namespace std;

int main() {

int\* arr;

int size;

cout << "Enter the size of the array: ";

cin >> size;

try {

arr = new int[size];

if (!arr) {

throw "Memory allocation failed.";

}

cout << "Memory allocation successful." << endl;

delete[] arr;

cout << "Memory deallocated successfully." << endl;

}

catch (const char\* errMsg) {

cerr << "Error: " << errMsg << endl;

exit(1);

}

return 0;

}

A screenshot of a computer

Description automatically generated

23.

#include <iostream>

#include <vector>

using namespace std;

class Matrix {

private:

vector<int> data;

size\_t rows;

size\_t cols;

public:

Matrix(size\_t rows, size\_t cols) : rows(rows), cols(cols) {

data.resize(rows, vector<int>(cols, 0));

}

Matrix operator+(const Matrix& other) const {

if (rows != other.rows || cols != other.cols) {

throw runtime\_error("Matrices must have the same dimensions for addition.");

}

Matrix result(rows, cols);

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

result.data[i][j] = data[i][j] + other.data[i][j];

}

}

return result;

}

Matrix operator-(const Matrix& other) const {

if (rows != other.rows || cols != other.cols) {

throw runtime\_error("Matrices must have the same dimensions for subtraction.");

}

Matrix result(rows, cols);

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

result.data[i][j] = data[i][j] - other.data[i][j];

}

}

return result;

}

Matrix operator\*(const Matrix& other) const {

if (cols != other.rows) {

throw runtime\_error("Number of columns of first matrix must be equal to the number of rows of second matrix for multiplication.");

}

Matrix result(rows, other.cols);

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < other.cols; ++j) {

for (size\_t k = 0; k < cols; ++k) {

result.data[i][j] += data[i][k] \* other.data[k][j];

}

}

}

return result;

}

friend ostream& operator<<(ostream& out, const Matrix& mat) {

for (size\_t i = 0; i < mat.rows; ++i) {

for (size\_t j = 0; j < mat.cols; ++j) {

out << mat.data[i][j] << ' ';

}

out << endl;

}

return out;

}

};

int main() {

Matrix A(2, 3);

A = { {1, 2, 3}, {4, 5, 6} };

Matrix B(2, 3);

B = { {7, 8, 9}, {10, 11, 12} };

cout << "Matrix A + Matrix B:" << endl;

cout << A + B << endl;

cout << "Matrix A - Matrix B:" << endl;

cout << A - B << endl;

Matrix C(3, 2);

C = { {1, 2}, {3, 4}, {5, 6} };

Matrix D(2, 2);

D = { {7, 8}, {9, 10} };

cout << "Matrix C \* Matrix D:" << endl;

cout << C \* D << endl;

return 0;

}

24.

#include <iostream>

#include <cmath>

using namespace std;

double calculateArea(double length, double width) {

return length \* width;

}

double calculateArea(double radius) {

return M\_PI \* radius \* radius;

}

double calculateArea(double base, float height) {

return 0.5 \* base \* height;

}

int main() {

double length, width, radius, base, height;

cout << "Enter the length and width of the rectangle: ";

cin >> length >> width;

cout << "Area of the rectangle: " << calculateArea(length, width) << endl;

cout << "Enter the radius of the circle: ";

cin >> radius;

cout << "Area of the circle: " << calculateArea(radius) << endl;

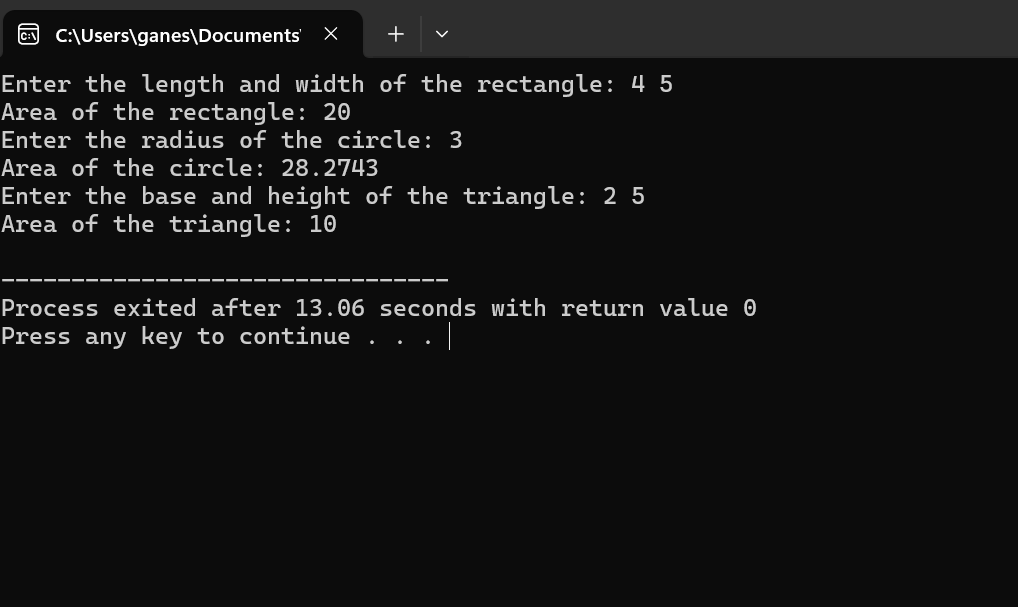
cout << "Enter the base and height of the triangle: ";

cin >> base >> height;

cout << "Area of the triangle: " << calculateArea(base, height) << endl;

return 0;

}



25.

#include <iostream>

#include <stack>

#include <string>

#include <sstream>

#include <cmath>

using namespace std;

bool isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

int precedence(char op) {

if (op == '+' || op == '-')

return 1;

if (op == '\*' || op == '/')

return 2;

if (op == '^')

return 3;

return 0;

}

double evaluatePostfix(const string& postfix) {

stack<double> operands;

for (char c : postfix) {

if (isdigit(c)) {

operands.push(c - '0');

} else if (isOperator(c)) {

double operand2 = operands.top();

operands.pop();

double operand1 = operands.top();

operands.pop();

double result;

switch (c) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

if (operand2 == 0) {

throw runtime\_error("Division by zero error.");

}

result = operand1 / operand2;

break;

case '^':

result = pow(operand1, operand2);

break;

}

operands.push(result);

}

}

return operands.top();

}

string infixToPostfix(const string& infix) {

stringstream postfix;

stack<char> operators;

for (char c : infix) {

if (isdigit(c)) {

postfix << c;

} else if (c == '(') {

operators.push(c);

} else if (c == ')') {

while (!operators.empty() && operators.top() != '(') {

postfix << operators.top();

operators.pop();

}

operators.pop(); // Discard the '('

} else if (isOperator(c)) {

while (!operators.empty() && precedence(operators.top()) >= precedence(c)) {

postfix << operators.top();

operators.pop();

}

operators.push(c);

}

}

while (!operators.empty()) {

postfix << operators.top();

operators.pop();

}

return postfix.str();

}

int main() {

string expression;

cout << "Enter a mathematical expression: ";

getline(cin, expression);

try {

string postfix = infixToPostfix(expression);

cout << "Postfix expression: " << postfix << endl;

double result = evaluatePostfix(postfix);

cout << "Result: " << result << endl;

} catch (const exception& e) {

cerr << "Error: " << e.what() << endl;

}

return 0;

}

26.

#include <iostream>

using namespace std;

template<typename T>

T findMax(const T arr[], int size) {

T maxElement = arr[0];

for (int i = 1; i < size; ++i) {

if (arr[i] > maxElement) {

maxElement = arr[i];

}

}

return maxElement;

}

int main() {

int intArray[] = {1, 3, 5, 2, 4};

int intMax = findMax(intArray, 5);

cout << "Maximum element in integer array: " << intMax << endl;

float floatArray[] = {1.1, 3.3, 5.5, 2.2, 4.4};

float floatMax = findMax(floatArray, 5);

cout << "Maximum element in float array: " << floatMax << endl;

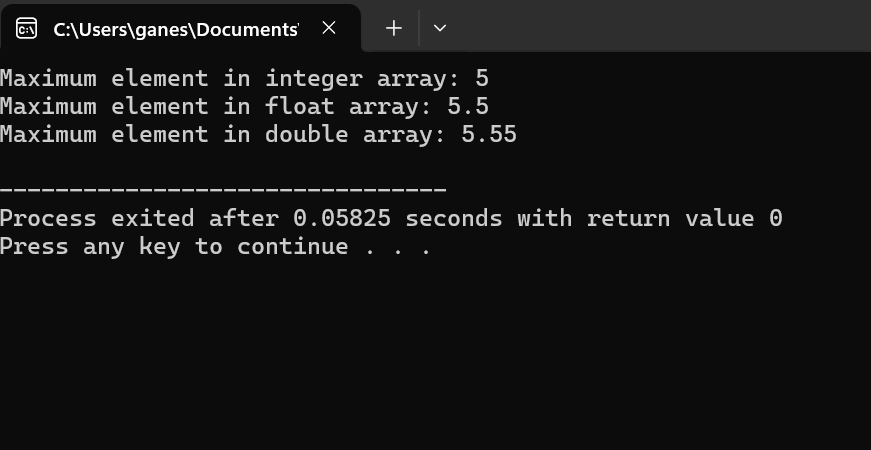
double doubleArray[] = {1.11, 3.33, 5.55, 2.22, 4.44};

double doubleMax = findMax(doubleArray, 5);

cout << "Maximum element in double array: " << doubleMax << endl;

return 0;

}



27.

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char source, char auxiliary, char destination) {

if (n == 1) {

cout << "Move disk 1 from rod " << source << " to rod " << destination << endl;

return;

}

towerOfHanoi(n - 1, source, destination, auxiliary);

cout << "Move disk " << n << " from rod " << source << " to rod " << destination << endl;

towerOfHanoi(n - 1, auxiliary, source, destination);

}

int main() {

int numDisks;

cout << "Enter the number of disks: ";

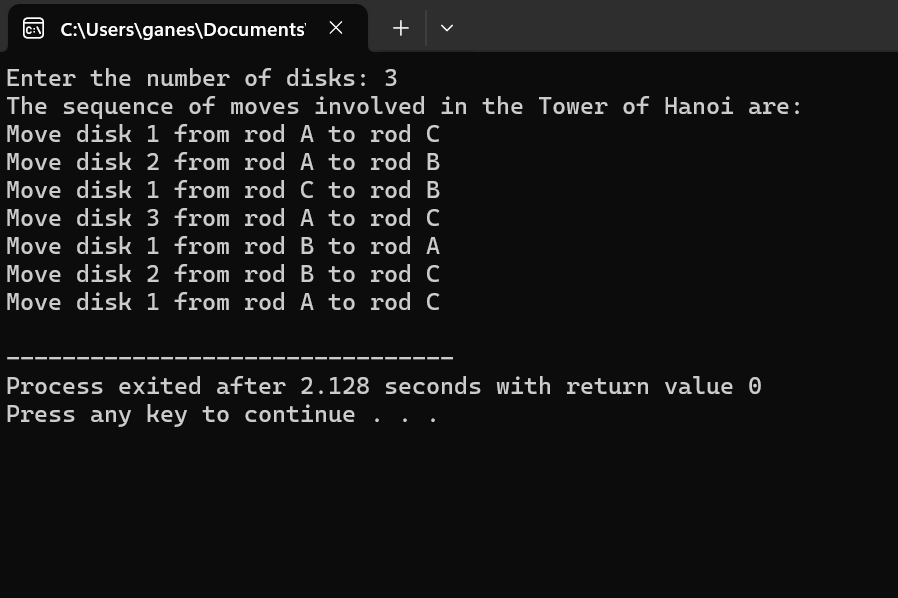
cin >> numDisks;

cout << "The sequence of moves involved in the Tower of Hanoi are:" << endl;

towerOfHanoi(numDisks, 'A', 'B', 'C');

return 0;

}



28.

#include <iostream>

#include <fstream>

#include <vector>

#include <stdexcept>

using namespace std;

double calculateAverage(const vector<double>& data) {

double sum = 0.0;

for (double value : data) {

sum += value;

}

return data.size() == 0 ? 0.0 : sum / data.size();

}

int main() {

string filename;

cout << "Enter the filename: ";

cin >> filename;

try {

ifstream file(filename);

if (!file) {

throw runtime\_error("Unable to open file.");

}

vector<double> data;

double value;

while (file >> value) {

data.push\_back(value);

}

if (file.eof()) {

double average = calculateAverage(data);

cout << "Average of the numbers: " << average << endl;

} else {

throw runtime\_error("Invalid data encountered.");

}

file.close();

} catch (const exception& e) {

cerr << "Error: " << e.what() << endl;

}

return 0;

}

29.

#include <iostream>

typedef void (\*CallbackFunction)(int);

void processCallback(CallbackFunction callback, int data) {

callback(data);

}

void callbackFunction1(int data) {

std::cout << "Callback function 1 called with data: " << data << std::endl;

}

void callbackFunction2(int data) {

std::cout << "Callback function 2 called with data: " << data << std::endl;

}

int main() {

int data = 42;

std::cout << "Calling processCallback with callbackFunction1:" << std::endl;

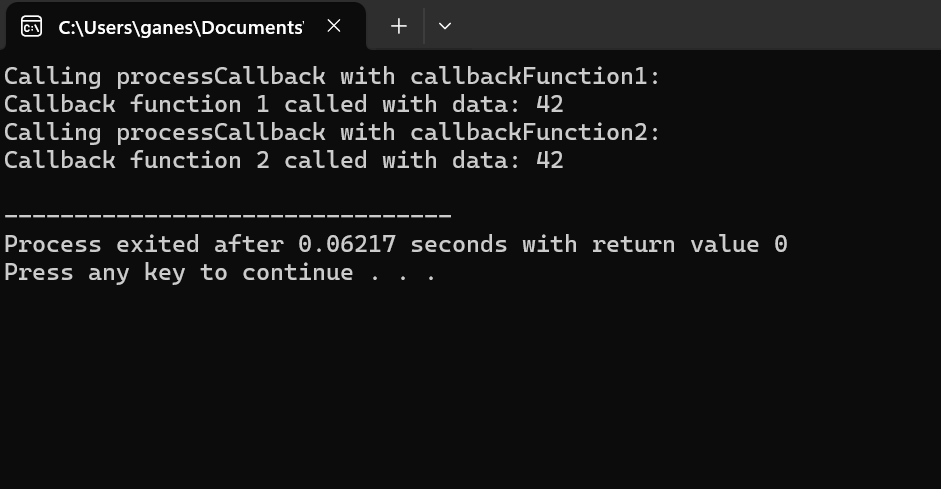
processCallback(callbackFunction1, data);

std::cout << "Calling processCallback with callbackFunction2:" << std::endl;

processCallback(callbackFunction2, data);

return 0;

}



30.

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

Node(int k) : key(k), left(nullptr), right(nullptr) {}

};

Node\* insert(Node\* root, int key) {

if (root == nullptr)

return new Node(key);

if (key < root->key)

root->left = insert(root->left, key);

else if (key > root->key)

root->right = insert(root->right, key);

return root;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current && current->left != nullptr)

current = current->left;

return current;

}

Node\* deleteNode(Node\* root, int key) {

if (root == nullptr)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == nullptr) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

cout << root->key << " ";

inorder(root->right);

}

}

void preorder(Node\* root) {

if (root != nullptr) {

cout << root->key << " ";

preorder(root->left);

preorder(root->right);

}

}

void postorder(Node\* root) {

if (root != nullptr) {

postorder(root->left);

postorder(root->right);

cout << root->key << " ";

}

}

int main() {

Node\* root = nullptr;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

cout << "In-order traversal: ";

inorder(root);

cout << endl;

cout << "Pre-order traversal: ";

preorder(root);

cout << endl;

cout << "Post-order traversal: ";

postorder(root);

cout << endl;

root = deleteNode(root, 20);

cout << "In-order traversal after deleting 20: ";

inorder(root);

cout << endl;

return 0;

}